java8 系列 -06 时间相关 API

作者：pleaseok
原文链接：https://ld246.com/article/1552147983470
来源网站：链滴

用过旧版java时间类库的都清楚，其实用起来的感觉是非常糟糕的。比如java.sql.Date与java.util.Dat除了包名以外类名都是一样的。(sql.Date是继承util.Date重写的)这种设计其实是非常差的，很容易混淆。再比如，当你在并发情景下用java.util.Date是很容易出问题的，因为java.util.Date是属于线程安全的。针对这些问题，java8推出了一系列新的API. PS: 新的时间相关API都在java.time.*包下

时间日期类

表示时间戳的instant

instant是用来表示时间戳的，等价于System.currentTimeMillis(),instant类被设计成单例模式，使起来也是极其方便的。

Instant类构建的几种方法以及他们的作用

```
public class InstantDemo {

    public static void main(String[] args) {
        Instant now = Instant.now(); // return Clock.systemUTC().instant(); 也就是说默认的是格林区的时间
        System.out.println(now.toEpochMilli());

        // 相对于格林时区偏移八小时，就是中国的上海时区啦
        Instant now2 = Instant.now(Clock.offset(Clock.systemUTC(), Duration.ofHours(8)));
        System.out.println(now2.toEpochMilli());

        Instant from = Instant.from(ZonedDateTime.now()); // 从时态对象(TemporalAccessor接口获取Instant的实例
        System.out.println(from.toEpochMilli());

        Instant milli = Instant.ofEpochMilli(System.currentTimeMillis()); // 通过毫秒数值直接构建Intant实例
```

```
        System.out.println(milli.getEpochSecond());

        Instant second = Instant.ofEpochSecond(60); // 通过传入一个标准时间的偏移值来构建Instant实例
        System.out.println(second);

        Instant second2 = Instant.ofEpochSecond(60,60); // 第二个参数:纳秒调整到秒数，正或负
        System.out.println(second2);

        System.out.println(System.currentTimeMillis());
    }
}
```

输出

```
1552128157932
1552156957932
1552128158057
1552128158
1970-01-01T00:01:00Z
1970-01-01T00:01:00.000000060Z
1552128158073
```

由上可知，Instant默认是基于格林时区的，它可以通过.toEpochMilli()来转换成跟System.currentTieMillis()一样效果的时间戳格式，当然它也可以直接通过.getEpochSecond()来直接转换成秒的时间格式

表示日期的LocalDate

以前的Date类在并发情境下做处理是很容易出现许多问题的（可以开多个线程来玩一下），而LocalDte可以更好的处理这种情况。从源码可以看到这一点: 如final class LocalDate,private LocalDate(){}..，有一点点并发基础的一定懂这些含义。

LocalDate类构建的几种方法以及他们的作用

```
public class LocalDateDemo {
    public static void main(String[] args) {
        LocalDate now = LocalDate.now(); // 和Instance一样默认的是格林时区
        System.out.println(now);

        LocalDate now2 = LocalDate.now(Clock.offset(Clock.systemUTC(), Duration.ofHours(24))); // 偏移格林时区24小时
        System.out.println(now2);

        LocalDate now3 = LocalDate.now(ZoneId.systemDefault()); // java.time.ZoneId是原有的java.util.TimeZone的替代品
        System.out.println(now3);

        LocalDate from = LocalDate.from(ZonedDateTime.now()); // 从时态对象(TemporalAccesso接口)获取LocalDate的实例
        System.out.println(from);

        LocalDate of = LocalDate.of(2017,5,21); // 参数分别为:year,month,dayOfMonth
        System.out.println(of);
```

```
        LocalDate epochDay = LocalDate.ofEpochDay(1800); // 比1970-01-01大多少天
        System.out.println(epochDay);

        LocalDate ofY = LocalDate.ofYearDay(2018,122); // 参数分别为 year , dayOfYear
        System.out.println(ofY);

        LocalDate parse = LocalDate.parse("2018-06-09"); // 字符串转换成LocalDate对象
        System.out.println(parse);
    }
}
```

输出

```
2019-03-09
2019-03-10
2019-03-09
2019-03-09
2017-05-21
1974-12-06
2018-05-02
2018-06-09
```

LocalDate类一些比较有用的方法

- public int getYear()：获取年份信息
- public int getMonthValue()：获取月份信息
- public int getDayOfMonth()：获取当前日是这个月的第几天
- public int getDayOfYear()：获取当前日是这一年的第几天
- public boolean isLeapYear()：是否是闰年
- public int lengthOfYear()：获取这一年有多少天
- public DayOfWeek getDayOfWeek()：返回星期信息
- public String format(DateTimeFormatter formatter): 格式化LocalDate
- public LocalDate minusYears(long yearsToSubtract)：返回此LocalDate对象，并减去指定的年。
- public LocalDate plusDays(long daysToAdd)：返回此LocalDate对象，并添加指定的天数。

太多了，这里就不一一列出来了...

时间LocalTime

LocalTime其实与上面的LocalDate类似，只不过LocalTime是来表示时间，而LocalDate是用作日期

LocalTime类构建

```
public class LocalTimeDemo {
    public static void main(String[] args) {
        LocalTime now = LocalTime.now();
        System.out.println(now); // 输出19:54:31.010
    }
```

}

- static LocalTime now()：在默认时区中从系统时钟获取当前时间。
- static LocalTime now(Clock clock)：从指定的时钟获得当前时间。
- static LocalTime now(ZoneId zone)：从指定时区的系统时钟获取当前时间。
- static LocalTime ofNanoOfDay(long nanoOfDay)：从一天的纳米值获得LocalTime的实例。
- static LocalTime from(TemporalAccessor temporal)：从temporal对象获取LocalTime的实例。
- static LocalTime of(int hour, int minute)：　从指定小时分钟获得LocalTime的实例。
- static LocalTime of(int hour, int minute, int second)：从指定小时，分钟和秒钟获取LocalTime实例。
- static LocalTime of(int hour, int minute, int second, int nanoOfSecond)：从指定小时，分钟秒和纳秒获得LocalTime的实例。
- static LocalTime parse(CharSequence text)：从文本字符串中获取LocalTime的实例。

localTime类的一些方法

```
public class LocalTimeDemo {
    public static void main(String[] args) {
        LocalTime now = LocalTime.now();
        String minute = now.getMinute()<10?"0"+now.getMinute():String.valueOf(now.getMinut
());
        String second = now.getSecond()<10?"0"+now.getSecond():String.valueOf(now.getSeco
d());
        System.out.println("当前时间:"+now.getHour()+":"+minute+":"+second); //当前时间:20:09
13
        // 获取纳秒字段的值。
        System.out.println(now.getNano()); // 434000000
        // 与另一个时间比较。
        System.out.println(now.compareTo(LocalTime.MAX)); // -1
        // 将日期装换成String
        System.out.println(now.toString()); // 20:09:13.434
        // 将此时间与日期相结合以创建LocalDateTime。
        System.out.println(now.atDate(LocalDate.now())); // 2019-03-09T20:09:13.434
        // 创建一个新的LocalTime，并更改日期。
        System.out.println(now.withHour(8)); // 08:09:13.434
        // 创建一个此时间的偏移时间的LocalTime
        System.out.println(now.atOffset(ZoneOffset.ofHours(8))); // 20:09:13.434+08:00
        // 创建一个此时间加上8小时的Localtime
        System.out.println(now.plusHours(8)); // 04:09:13.434
    }
}
```

关于LocalTime还有一些其他的方法，请自行研究~~

时区相关的日期时间ZonedDateTime

上面讲的LocalDate、LocalTime还有未讲到的LocalDateTime都是与时区无关的，都只是用的默认区，而类的本身并没有存储时区。而上面的类要用到时区的话，还需要此类来协助。

```
public class ZonedDateTimeDemo {
```

```
    public static void main(String[] args) {
        ZonedDateTime zdt = ZonedDateTime.now();
        System.out.println(zdt); // 2019-03-09T23:14:48.560+08:00[Asia/Shanghai]

        ZonedDateTime zdt2 = ZonedDateTime.of(LocalDate.now(), LocalTime.now(), ZoneId.of(
America/Chicago"));
        System.out.println(zdt2); // 2019-03-09T23:14:48.560-06:00[America/Chicago]

        ZonedDateTime zdt3 = ZonedDateTime.ofInstant(Instant.now(),ZoneId.of("Africa/Cairo"))

        System.out.println(zdt3); // 2019-03-09T17:14:48.591+02:00[Africa/Cairo]
    }
}
```

解释一下，第一个输出系统默认时区。第二个输出的是没过芝加哥时区，两个参数，第一个是LocalDa
e，第二个是LocalTime(上面说过这两个类默认是没有任何时区的，想要时区还的结合ZonedDateTi
e类)。第二个输出是非洲开罗时区，这里传入的参数是时间戳Instant

关于时间的工具类

日期时间格式化

上面讲LocalDate的时候有讲到一个format方法，里面传入的参数是DateTimeFormatter，那这里就
一下这个类吧。其实也没什么很多要讲的，毕竟它用起来真的太方便啦。一起来看看

```
public class DateTimeFormatterDemo {

    public static void main(String[] args) {
        LocalDateTime currentTime = LocalDateTime.now();

        // DateTimeFormatter默认规范有很多，这里只列举出了三个
        System.out.println(currentTime.format(DateTimeFormatter.BASIC_ISO_DATE)); // 201903
9

        System.out.println(currentTime.format(DateTimeFormatter.ISO_DATE_TIME)); // 2019-03
09T23:39:02.261

        System.out.println(currentTime.format(DateTimeFormatter.ISO_LOCAL_DATE)); // 2019-0
-09

        System.out.println(currentTime.format(DateTimeFormatter.ofPattern("yyyyMMdd"))); //
0190309
    }
}
```

真的使用起来太简单了，并且你想要的格式它基本都有

时间差 Period\Duration

1. Period

Period是以年月日来衡量一个时间段，比如几年几月几日

```
public class PeriodDemo {

    public static void main(String[] args) {
        Period period = Period.of(2018,8,22);
        System.out.println(period);

        System.out.println(Period.between(LocalDate.of(2018,3,5), LocalDate.now()));
    }
}
```

输出

```
P2018Y8M22D
P1Y4D
```

- P:标识开头
- Y:年
- M:月
- D:日

2. Duration

Duration也是表示一个时间段。不知道各位有没有注意到，其实上面例子中有用到过这个类 Duration ofHours(8) 向后推迟8小时

```
public class DurationDemo {
    public static void main(String[] args) {
        LocalDateTime from = LocalDateTime.of(2018,Month.DECEMBER,22,22,33);//2018-12-22 22:33
        LocalDateTime to = LocalDateTime.now(); // 2019-03-10T00:05:42.443
        Duration duration = Duration.between(from, to); // PT1849H33M20.732S

        long days = duration.toDays();              // 这段时间的总天数
        long hours = duration.toHours();            // 这段时间的小时数
        long minutes = duration.toMinutes();        // 这段时间的分钟数
        long seconds = duration.getSeconds();       // 这段时间的秒数
        long milliSeconds = duration.toMillis();    // 这段时间的毫秒数
        long nanoSeconds = duration.toNanos();      // 这段时间的纳秒数
    }
}
```

其实使用多了这些类你会发现，这些类都有许多的共用。这些我这里就没有总结，请多动手，习惯成然。